

# Chapter 4

## Router Configuration

This chapter explains how to use the JUNOScript application programming interface (API) to change router configuration or to request information about the current configuration. The JUNOScript tags described here correspond to JUNOS command-line interface (CLI) configuration statements described in the JUNOS Internet software configuration guides.

This chapter discusses the following topics:

- [Mapping between CLI Configuration Statements and JUNOScript Tags on page 40](#)
- [Same Tags Used for Requests and Responses on page 45](#)
- [Overview of Router Configuration Procedures on page 46](#)
- [Lock the Candidate Configuration on page 47](#)
- [Request Configuration Information on page 48](#)
- [Change the Candidate Configuration on page 55](#)
- [Verify the Syntactic Correctness of the Candidate Configuration on page 65](#)
- [Commit the Candidate Configuration on page 66](#)
- [Unlock the Candidate Configuration on page 68](#)

- **Mapping between CLI Configuration Statements and JUNOScript Tags**

The JUNOScript API defines a tag for every container and leaf statement in the JUNOS configuration hierarchy. At the top levels of the configuration hierarchy, there is almost always a one-to-one mapping between tags and statements, and most tag names match the configuration statement name. At deeper levels of the configuration hierarchy, the mapping is sometimes less direct, because some CLI notational conventions do not map directly to Extensible Markup Language (XML)-compliant tagging syntax. The following sections describe the mapping between configuration statements and JUNOScript tags:

- Tag Mappings for Top-Level (Container) Statements on page 40

- Tag Mappings for Leaf Statements on page 41

- Tag Mappings for Identifiers on page 42

- Tag Mappings for Leaf Statements with Multiple Values on page 43

- Tag Mappings for Multiple Options on One or More Lines on page 44



For some configuration statements, the notation used when typing the statement at the CLI configuration-mode prompt differs from the notation used in a configuration file. The same JUNOScript tag maps to both notational styles.

## **Tag Mappings for Top-Level (Container) Statements**

The <configuration> tag is the top-level JUNOScript container tag for configuration statements, and corresponds to the CLI [edit] level. Most statements at the next few levels of the configuration hierarchy are container statements, and the name of each corresponding JUNOScript container tag almost always matches the container statement name.

(The top-level <configuration-text> tag also corresponds to the CLI configuration mode's [edit] level. It encloses formatted ASCII configuration statements instead of JUNOScript tags, and is not relevant to the following discussion. For more information, see “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 49.)

The following depicts the mappings for two sample statements that begin at the top level of the configuration hierarchy. Note how a closing brace in a CLI configuration statement corresponds to a closing JUNOScript tag:

### **CLI Configuration Statements**

```
system {
    login {
        ...child statements...
    }
}
protocols {
    ospf {
        ...child statements...
    }
}
```

### **JUNOScript Tags**

```
<configuration>
    <system>
        <login>
            ...child statements - -
        </login>
    </system>
    <protocols>
        <ospf>
            ...child statements - -
        </ospf>
    </protocols>
</configuration>
```

## Tag Mappings for Leaf Statements

A *leaf statement* is a CLI configuration statement that does not contain any other statements. Most leaf statements define a value for one characteristic of a configuration object and have the following form:

```
keyword value;
```

In general, the name of the JUNOScript tag corresponding to a leaf statement is the same as the keyword string. The string between the opening and closing JUNOScript tags is the *value*.

The following depicts the mappings for two sample leaf statements that have a keyword and a value: the message statement at the [edit system login] level and the preference statement at the [edit protocols ospf] level:

### CLI Configuration Statements

```
system {
    login {
        message "Authorized users only";
        ...other statements under login ...
    }
}
protocols {
    ospf {
        preference 15;
        ...other statements under ospf ...
    }
}
```

### JUNOScript Tags

```
<configuration>
    <system>
        <login>
            <message>Authorized users only</message>
            <!-- other children of the <login> tag -->
        </login>
    </system>
    <protocols>
        <ospf>
            <preference>15</preference>
            <!-- other children of the <ospf> tag -->
        </ospf>
    </protocols>
</configuration>
```

T1010

Some leaf statements consist of a fixed-form keyword only, without an associated variable-form value. The JUNOScript API represents such statements with an empty tag. The following example shows the mapping for the disable statement at the [edit forwarding-options sampling] hierarchy level:

### CLI Configuration Statement

```
forwarding-options {
    sampling {
        disable;
        ...other statements under sampling ...
    }
}
```

### JUNOScript Tags

```
<configuration>
    <forwarding-options>
        <sampling>
            <disable/>
            <!-- other children of the <sampling> tag -->
        </sampling>
    </forwarding-options>
</configuration>
```

T1011

## Tag Mappings for Identifiers

At some hierarchy levels, the same kind of container object can appear multiple times. Each instance of the object has a unique identifier to distinguish it from the other instances. In the JUNOS CLI notation, the first statement in the set of statements for such an object consists of a keyword and identifier of the following form:

```
keyword identifier {  
    ... configuration statements for individual characteristics ...  
}
```

The keyword is a fixed string that indicates the type of object being defined, and the *identifier* is the unique name for this instance of the type. In the JUNOScript API, the tag corresponding to the keyword is a container tag for child tags that represent the object's characteristics. The container tag's name generally matches the keyword string.

The JUNOScript API differs from the CLI in its treatment of the identifier. Because the JUNOScript API does not allow container tags to contain both other tags and untagged character data such as an identifier name, the identifier must be enclosed in a tag of its own. Most frequently, identifier tags are called <name>. To verify the identifier tag name for an object, consult the object's entry in the appropriate document type definition (DTD) or in the *JUNOScript API Reference*.

Identifier tags also constitute an exception to the general XML convention that tags at the same level of hierarchy can appear in any order; the identifier tag always occurs first within the container tag.

The [edit protocols bgp group] statement is an example of a configuration statement with an identifier. Each Border Gateway Protocol (BGP) group has an associated name (the identifier) and can have other characteristics such as a type, peer autonomous system (AS) number, and neighbor address.

The following example illustrates the mapping between the CLI statements and JUNOScript tags that create two BGP groups called G1 and G2. Notice how the JUNOScript <name> tag that encloses the identifier for each group (and for the identifier of the neighbor within a group) does not have a counterpart in the CLI statements. For complete information about changing router configuration, see “Change the Candidate Configuration” on page 55.

CLI Configuration Statements	JUNOScript Tags
<pre>protocols {     bgp {         group G1 {             type external;             peer-as 56;             neighbor 10.0.0.1;         }         group G2 {             type external;             peer-as 57;             neighbor 10.0.10.1;         }     } }</pre>	<pre>&lt;configuration&gt;     &lt;protocols&gt;         &lt;bgp&gt;             &lt;group&gt;                 &lt;name&gt;G1&lt;/name&gt;                 &lt;type&gt;external&lt;/type&gt;                 &lt;peer-as&gt;56&lt;/peer-as&gt;                 &lt;neighbor&gt;                     &lt;name&gt;10.0.0.1&lt;/name&gt;                 &lt;/neighbor&gt;             &lt;/group&gt;             &lt;group&gt;                 &lt;name&gt;G2&lt;/name&gt;                 &lt;type&gt;external&lt;/type&gt;                 &lt;peer-as&gt;57&lt;/peer-as&gt;                 &lt;neighbor&gt;                     &lt;name&gt;10.0.10.1&lt;/name&gt;                 &lt;/neighbor&gt;             &lt;/group&gt;         &lt;/bgp&gt;     &lt;/protocols&gt; &lt;/configuration&gt;</pre>

T1012

## Tag Mappings for Leaf Statements with Multiple Values

Some JUNOS CLI leaf statements accept multiple values, which can either be user-defined or drawn from a set of predefined values. CLI notation uses square brackets to enclose all values in a single statement, as in the following:

*statement [ value1 value2 value3 ];*

The JUNOScript API instead encloses each value in its own tag. The following example illustrates the mapping between a CLI statement with multiple user-defined values and the corresponding JUNOScript tags. The import statement imports two routing policies defined elsewhere in the configuration. For complete information about changing router configuration, see “Change the Candidate Configuration” on page 55.

CLI Configuration Statements	JUNOScript Tags
<pre>protocols {     bgp {         group 23 {             import [ policy1 policy2 ];         }     } }</pre>	<pre>&lt;configuration&gt;     &lt;protocols&gt;         &lt;bgp&gt;             &lt;group&gt;                 &lt;name&gt;23&lt;/name&gt;                 &lt;import&gt;policy1&lt;/import&gt;                 &lt;import&gt;policy2&lt;/import&gt;             &lt;/group&gt;         &lt;/bgp&gt;     &lt;/protocols&gt; &lt;/configuration&gt;</pre>

T1013

- The following example illustrates the same mapping for a CLI statement with multiple predefined values. The permissions statement grants three predefined JUNOS permissions to members of the user-accounts login class.

**CLI Configuration Statements**

```
system {
    login {
        class user-accounts {
            permissions [ configure admin control ];
        }
    }
}
```

**JUNOScript Tags**

```
<configuration>
    <system>
        <login>
            <class>
                <name>user-accounts</name>
                <permissions>configure</permissions>
                <permissions>admin</permissions>
                <permissions>control</permissions>
            </class>
        </login>
    </system>
</configuration>
```

T1014

***Tag Mappings for Multiple Options on One or More Lines***

For some configuration objects, the configuration file specifies the value for more than one option on a single line, usually for greater legibility and conciseness. In most such cases, the first option identifies the object and does not have a keyword, but later options are paired keywords and values. The JUNOScript API encloses each option in its own tag. Because the first option has no keyword in the CLI statement, the JUNOScript API assigns a tag name to it.

The following example illustrates the mapping of a CLI configuration statement that places multiple options on a single line to the corresponding JUNOScript tags. Notice that the JUNOScript API defines a tag for both options and assigns a tag name to the first option (10.0.0.1), which has no CLI keyword.

**CLI Configuration Statements**

```
system {
    backup-router 10.0.0.1 destination 10.0.0.2;
}
```

**JUNOScript Tags**

```
<configuration>
    <system>
        <backup-router>
            <address>10.0.0.1</address>
            <destination>10.0.0.2</destination>
        </backup-router>
    </system>
</configuration>
```

T1015

The configuration file entries for some configuration objects have multiple lines with more than one option, and again the JUNOScript API defines a separate tag for each option. The following example illustrates the mappings for a sample [edit protocols isis traceoptions] statement, which contains three statements, each with multiple options:

### CLI Configuration Statements

```
protocols {
    isis {
        traceoptions {
            file trace-file size 3m files 10 world-readable;
            flag route detail;
            flag state receive;
        }
    }
}
```

### JUNOScript Tags

```
<configuration>
    <protocols>
        <isis>
            <traceoptions>
                <file>
                    <filename>trace-file</filename>
                    <size>3m</size>
                    <files>10</files>
                    <world-readable/>
                </file>
                <flag>
                    <name>route</name>
                    <detail/>
                </flag>
                <flag>
                    <name>state</name>
                    <receive/>
                </flag>
            </traceoptions>
        </isis>
    </protocols>
</configuration>
```

T1016

## Same Tags Used for Requests and Responses

The JUNOScript server encloses its response to a configuration request in `<rpc-reply>` tags, just as for an operational request. Within the `<rpc-reply>` tags, it by default returns a JUNOScript-tagged response enclosed in `<configuration>` tags. (If the client application requests a formatted ASCII response, the server instead encloses the response in `<configuration-text>` tags. For more information, see “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 49.)

Enclosing every JUNOScript-tagged configuration response within `<configuration>` tags contrasts with how the server encloses each different kind of operational response in a tag named for that type of response—for example, the `<chassis-inventory>` tag for chassis information or the `<interface-information>` tag for interface information.

The JUNOScript tags within the `<configuration>` tags represent configuration hierarchy levels, configuration objects, and object characteristics, always ordered from higher to deeper levels of the hierarchy. When a client application loads a configuration, it emits the same tags in the same order as the JUNOScript server uses when returning configuration information. This consistent representation helps make handling configuration information more straightforward. For instance, the client application can change router configuration by requesting the current configuration, storing the JUNOScript server’s response to a local memory buffer, making changes or applying transformations to the buffered data, and reloading the altered configuration. Because the altered configuration is based on the JUNOScript server’s response, it is certain to be syntactically correct. For more information about changing router configuration, see “Change the Candidate Configuration” on page 55.

Similarly, when a client application requests information about a configuration hierarchy level or object, it uses the same tags that the JUNOScript server will return in response. To represent the hierarchy level or object about which it is requesting information, the client application sends a complete stream of tags from the top of the configuration hierarchy (represented by the <configuration> tag) down to the requested level or object. The innermost tag, which represents the level or object, is either empty or includes the identifier tag only. The JUNOScript server's response includes the same stream of tags, but the tag for the requested level or object contains all the tags that represent the level's child tags or object's characteristics. For more information about requesting configuration information, see “Request Configuration Information” on page 48.

One potential difference between the tag stream in the JUNOScript server's response and one emitted by a client application concerns the use of white space. By XML convention, the JUNOScript server ignores white space in the tag stream it receives. In the stream that it emits, however, the JUNOScript server includes newline characters and extra spaces between tags. If a client application writes the response to a file, each tag appears on its own line, and child tags are indented from their parents, both of which enhance readability for users. Client applications can ignore or discard the white space, particularly if they do not write the tags to a file for later review. Client applications do not need to include the white space in requests to the JUNOScript server.

## Overview of Router Configuration Procedures

To read or change router configuration information, perform the following procedures, which are described in the indicated sections:

1. Establish a connection to the JUNOScript server on the router, as described in “Prerequisites for telnet Connections” on page 17.
2. Open a JUNOScript session, as described in “Start the JUNOScript Session” on page 21.
3. (Optional) Lock the current candidate configuration, as described in “Lock the Candidate Configuration” on page 47. Locking the configuration prevents other users or applications from changing it at the same time.
4. Request or change configuration information, as described in “Request Configuration Information” on page 48 and “Change the Candidate Configuration” on page 55.
5. (Optional) Verify the syntactic correctness of the candidate configuration before attempting to commit it, as described in “Verify the Syntactic Correctness of the Candidate Configuration” on page 65.
6. Commit changes made to the candidate configuration (if appropriate), as described in “Commit the Candidate Configuration” on page 66.
7. Unlock the current configuration if it is locked, as described in “Unlock the Candidate Configuration” on page 68.
8. End the JUNOScript session and close the connection to the router, as described in “End the Session and Close the Connection” on page 29.

## Lock the Candidate Configuration

Before reading or changing router configuration, a client application must establish a connection to the JUNOScript server, as described in “Prerequisites for telnet Connections” on page 17, and open a JUNOScript session, as described in “Start the JUNOScript Session” on page 21.

The application can then emit the `<lock-configuration/>` tag enclosed in `<rpc>` tags if it needs to prevent other users or applications from changing the candidate configuration. If it is not important to block changes from other applications, the application can begin requesting or changing configuration information immediately, as described in “Request Configuration Information” on page 48 and “Change the Candidate Configuration” on page 55.

Only one application can hold the lock on the candidate configuration at a time. Other users and applications can still read the configuration while it is locked. The lock persists until either the JUNOScript session ends or the client application emits the `<unlock-configuration/>` tag, as described in “Unlock the Candidate Configuration” on page 68.

If the JUNOScript server successfully locks the configuration, it returns an opening `<rpc-reply>` and closing `</rpc-reply>` tag with nothing between them. If it cannot lock the configuration, it returns an `<xnm:error>` tag instead. Reasons for the failure include the following:

Another user or application has already locked the candidate configuration. The error message reports the login identity of the user or application.

The candidate configuration includes changes that have not yet been committed. To commit the changes, see “Commit the Candidate Configuration” on page 66. To roll back to a previous version of the configuration (and lose the uncommitted changes), see “Roll Back to a Previous Configuration” on page 65.

In the following example, the client application emits the `<lock-configuration/>` tag after opening the JUNOScript session and exchanging initialization information with the JUNOScript server:

<b>Client Application</b>	<b>JUNOScript Server</b>
<code>&lt;rpc&gt;</code> <code>  &lt;lock-configuration/&gt;</code> <code>&lt;/rpc&gt;</code>	<code>&lt;rpc-reply&gt;&lt;/rpc-reply&gt;</code>

T1003

## Automatically Discard Uncommitted Changes

By default, if a client application locks the configuration and does not commit it before the JUNOScript session ends, any uncommitted changes that are made during the session are retained in the candidate configuration. When the candidate configuration is subsequently committed, the leftover changes become part of the committed configuration. This can lead to unexpected results if the user or application performing the subsequent commit is unaware of the leftover changes.

To discard uncommitted changes from the candidate configuration when the JUNOScript session ends before the client application commits the configuration, enclose the `<rollback>` tag within the `<lock-configuration>` tag and set the `<rollback>` tag's value to automatic.

The following example illustrates the sequence of tags that causes an automatic rollback:

Client Application	JUNOScript Server
<pre>&lt;rpc&gt;   &lt;lock-configuration&gt;     &lt;rollback&gt;automatic&lt;/rollback&gt;   &lt;/lock-configuration&gt; &lt;/rpc&gt;</pre>	<pre>&lt;rpc-reply&gt;&lt;/rpc-reply&gt;</pre>

T1017

## Request Configuration Information

To request and parse configuration information, a client application emits the `<get-configuration>` tag. By setting optional attributes on the `<get-configuration>` tag and enclosing the appropriate child tags within it, the client application can request either the candidate or the committed configuration, specify either JUNOScript-tagged or formatted ASCII output, and request the entire configuration or just a section of it. The following sections describe the procedures for requesting configuration information:

[Specify the Committed or Candidate Configuration on page 49](#)

[Specify Formatted ASCII or JUNOScript-Tagged Output on page 49](#)

[Request the Complete Configuration on page 51](#)

[Request One Hierarchy Level on page 51](#)

[Request a Single Configuration Object on page 52](#)

If the client application has locked the candidate configuration as described in “Lock the Candidate Configuration” on page 47, it should unlock it after making its read requests. Other users and applications cannot change the configuration while it remains locked. For more information, see “Unlock the Candidate Configuration” on page 68.

Client applications can also request an XML schema representation of the complete hierarchy of JUNOS configuration statements, as described in the following section:

[Request an XML Schema for the Configuration Hierarchy on page 53](#)

## Specify the Committed or Candidate Configuration

To request information from the current committed configuration—the one active on the router—set the database attribute on the <get-configuration> tag to the value committed. To request information from the current candidate configuration, either set the database attribute to the value candidate or omit the database attribute completely (the JUNOScript server returns information from the candidate configuration by default). In either case, enclose the <get-configuration> tag in <rpc> tags.

The database attribute can be combined with the format attribute (described in “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 49), and included when requesting either the entire configuration or sections of it (as described in “Request the Complete Configuration” on page 51, “Request One Hierarchy Level” on page 51, and “Request a Single Configuration Object” on page 52).

The following example uses the database attribute to request the entire committed configuration:

<b>Client Application</b>	<b>JUNOScript Server</b>
<rpc>	
<get-configuration database="committed"/>	
</rpc>	
	<rpc-reply>
	<configuration>
	<version>5.3R1</version>
	<system>
	<host-name>big-router</host-name>
	<! - other children of the <system> tag - ->
	</system>
	<! - other children of the <configuration> tag - ->
	</configuration>
	</rpc-reply>

T1018

## Specify Formatted ASCII or JUNOScript-Tagged Output

To request that the JUNOScript server return configuration data as formatted ASCII text rather than tagged with JUNOScript tags, set the format attribute on the <get-configuration> tag to the value text. The server formats its response in the same way as the JUNOS CLI show configuration command displays configuration data—it uses the newline character, tabs, braces, and square brackets to indicate the hierarchical relationships between configuration statements. The server encloses formatted ASCII configuration information in <configuration-text> tags.

To request JUNOScript-tagged output, either set the format attribute to the value xml or omit the format attribute completely (the JUNOScript server returns JUNOScript-tagged output by default). The JUNOScript server encloses its output in <configuration> tags.

When the JUNOScript server encloses a JUNOScript tag in <undocumented> tags, the corresponding configuration element (hierarchy level or object) is not documented in the JUNOS software configuration guides or officially supported by Juniper Networks. Most often, the undocumented element is used for debugging purposes only by Juniper Networks personnel. In rarer cases, the element is no longer supported or has been moved to another area of the configuration hierarchy, but appears in the current location for backward compatibility.

Regardless of whether they are requesting JUNOScript tags for formatted ASCII, client applications must use JUNOScript tags to represent the configuration element to display. The format attribute controls the format of only the JUNOScript server's output. Enclose the tags that represent the element to display in <get-configuration> tags within <rpc> tags.

The format attribute can be combined with the database attribute (described in “Specify the Committed or Candidate Configuration” on page 49), and included when requesting either the entire configuration or sections of it (as described in “Request the Complete Configuration” on page 51, “Request One Hierarchy Level” on page 51, and “Request a Single Configuration Object” on page 52).

The following example uses the format attribute to request ASCII-formatted output at the [edit policy-options] level of the current candidate configuration:

<b>Client Application</b>	<b>JUNOScript Server</b>
<rpc>	
<get-configuration format="text">	
<configuration>	
<policy-options/>	
</configuration>	
</get-configuration>	
</rpc>	
	<rpc-reply>
	<configuration-text>
	policy-options {
	policy-statement load-balancing-policy {
	from {
	route-filter 192.168.10/24 orlonger;
	route-filter 9.114/16 orlonger;
	}
	then {
	load-balance per-packet;
	}
	}
	</configuration-text>
	</rpc-reply>

T1019

## **Request the Complete Configuration**

To request the entire current committed or candidate configuration, emit the empty <get-configuration/> tag enclosed in <rpc> tags. If desired, include the database or format attribute (or both), as described in “Specify the Committed or Candidate Configuration” on page 49 and “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 49, respectively.

The following example illustrates the tag sequence for requesting the current candidate configuration tagged with JUNOScript tags (the default):

### **Client Application JUNOScript Server**

```
<rpc>
    <get-configuration/>
</rpc>
<rpc-reply>
    <configuration>
        <version>5.3R1</version>
        <system>
            <host-name>big-router</host-name>
            <!- - other children of the <system> tag - ->
        </system>
        <!- - other children of the <configuration> tag - ->
    </configuration>
</rpc-reply>
```

T1020

## **Request One Hierarchy Level**

To request a single hierarchy level from the current committed or candidate configuration, emit <get-configuration> tags that enclose the tags representing all levels of the configuration hierarchy from the root (represented by the <configuration> tag) down to the level to display. Use an empty tag to represent the requested level. Enclose the entire request in <rpc> tags.

If desired, include the database or format attribute (or both), as described in “Specify the Committed or Candidate Configuration” on page 49 and “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 49, respectively. Note, however, that the format attribute controls the format of only the JUNOScript server’s output. Client applications must emit JUNOScript tags rather than formatted ASCII to represent which configuration level to display.

- The following example illustrates the tag sequence when a client application requests the contents of the [edit system login] level of the current candidate configuration. The output is tagged with JUNOScript tags (the default):

**Client Application JUNOScript Server**

```
<rpc>
    <get-configuration>
        <configuration>
            <system>
                <login/>
            </system>
        </configuration>
    </get-configuration>
</rpc>

<rpc-reply>
    <configuration>
        <system>
            <login>
                <user>
                    <name>barbara</name>
                    <full-name>Barbara Anderson</full-name>
                    <!-- other child tags for this user -->
                </user>
                <!-- other children of the <login> tag -->
            </login>
        </system>
    </configuration>
</rpc-reply>
```

T1021

## **Request a Single Configuration Object**

To request information about a single object at a specific level of the configuration hierarchy, emit `<get-configuration>` tags that enclose the tags representing the entire configuration hierarchy down to the object to display. To represent the requested object, emit its container tag and identifier tag only, not any tags that represent other characteristics.

If desired, include the database or format attribute (or both), as described in “Specify the Committed or Candidate Configuration” on page 49 and “Specify Formatted ASCII or JUNOScript-Tagged Output” on page 49, respectively. Note, however, that the format attribute controls the format of only the JUNOScript server’s output. Client applications must emit JUNOScript tags rather than formatted ASCII to represent which configuration object to display.

The following example illustrates the tag sequence when a client application requests the contents of one multicasting scope called local, which is at the [edit routing-options multicast] hierarchy level. To specify the configuration object about which to supply information, the client application emits the <name>local</name> identifier tag as the innermost tag. The output is from the current candidate configuration and is tagged with JUNOScript tags (the default).

<b>Client Application</b>	<b>JUNOScript Server</b>
<rpc>	
<get-configuration>	
<configuration>	
<routing-options>	
<multicast>	
<scope>	
<name>local</name>	
</scope>	
</multicast>	
</routing-options>	
</configuration>	
</get-configuration>	
</rpc>	
	<rpc-reply>
	<configuration>
	<routing-options>
	<multicast>
	<scope>
	<name>local</name>
	<prefix>239.255.0.0/16</prefix>
	<interface>ip-f/p/0</interface>
	</scope>
	</multicast>
	</routing-options>
	</configuration>
	</rpc-reply>

T1022

## **Request an XML Schema for the Configuration Hierarchy**

To request an XML Schema-language representation of the entire JUNOS configuration hierarchy, emit <get-xnm-information> tags that enclose the following two child tags with the indicated values:

<type>xml-schema</type>, which specifies that the JUNOScript server return the configuration as an XML schema

<namespace>junos-configuration</namespace>, which specifies that the JUNOScript server return information about the JUNOS configuration

The JUNOScript server returns the XML schema enclosed in <rpc-reply> and <xsd:schema> tags. The JUNOScript configuration schema represents the entire set of elements that can be configured on a Juniper Networks router that is running the version of the JUNOS software specified by the xmlns:junos attribute in the opening <rpc-reply> tag. Client applications can use the schema to validate the candidate or committed schema on a router, or to discover which configuration statements are available in that version of the JUNOS software.

- The JUNOScript configuration schema does not indicate which elements are actually configured on the router where the JUNOScript server is running, or even that an element can be configured on that type of router (some configuration statements are available only on certain router types). To request the set of currently configured elements and their settings, emit the <get-configuration> tag instead, as described in other sections in this chapter.

Explaining the structure and notational conventions of the XML Schema language is beyond the scope of this document. For a basic introduction to the XML Schema language, see the primer available at <http://www.w3.org/TR/xmlschema-0>. The primer references the more formal specifications for the XML Schema language if you need additional details.

The following examples illustrate the tag sequence with which a client application requests the JUNOScript configuration schema. In the JUNOScript server's reply, the first two `<xsd:element>` statements define the schema for the `<undocumented>` and `<comment>` JUNOScript tags, which can be enclosed in most other container tags defined in the schema (container tags are defined as `<xsd:complexType>`).

Client Application	JUNOScript Server
<pre>&lt;rpc&gt;   &lt;get-xnm-information&gt;     &lt;type&gt;       xml-schema     &lt;/type&gt;     &lt;namespace&gt;       junos-configuration     &lt;/namespace&gt;   &lt;/rpc&gt;</pre>	<pre>&lt;rpc-reply&gt; &lt;xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" \               elementFormDefault="qualified"&gt;   &lt;xsd:element name="undocumented"&gt;     &lt;xsd:complexType&gt;       &lt;xsd:sequence&gt;         &lt;xsd:any namespace="##any" processContents="skip"/&gt;       &lt;/xsd:sequence&gt;     &lt;/xsd:complexType&gt;   &lt;/xsd:element&gt;   &lt;xsd:element name="comment"&gt;     &lt;xsd:complexType&gt;       &lt;xsd:sequence&gt;         &lt;xsd:any namespace="##any" processContents="skip"/&gt;       &lt;/xsd:sequence&gt;     &lt;/xsd:complexType&gt;   &lt;/xsd:element&gt;</pre>

T1023

The third <xsd:element> statement in the schema defines the JUNOScript <configuration> tag. The fourth <xsd:element> statement begins the definition of the <system> tag, which corresponds to the [edit system] level of the JUNOScript configuration hierarchy. The statements corresponding to other hierarchy levels are omitted for brevity.

## Client Application JUNOScript Server

```
</xsd:element>
<xsd:element name="configuration">
<xsd:complexType>
<xsd:sequence>
<xsd:choice minOccurs="0" maxOccurs="unbounded">
<xsd:element ref="undocumented"/>
<xsd:element ref="comment"/>
<xsd:element name="system" minOccurs="0">
<xsd:complexType>
<xsd:sequence>
<xsd:choice minOccurs="0" maxOccurs="unbounded">
<xsd:element ref="undocumented"/>
<xsd:element ref="comment"/>
<! - child elements of <system> appear here - ->
</xsd:choice >
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<! - statements for other hierarchy levels appear here - ->
</xsd:choice >
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>
</rpc-reply>
```

T1024

## Change the Candidate Configuration

To change the current candidate configuration, emit the <load-configuration> tag. Specify which configuration element (hierarchy level or configuration object) to change in one of two ways:

By setting the empty <load-configuration> tag's url attribute to the pathname of a file that resides on the router and contains the set of configuration statements to load. For example, the following tag identifies the file /tmp/new.conf as the file to load:

```
<load-configuration url="/tmp/new.conf"/>
```

By enclosing the appropriate configuration statements within opening and closing <load-configuration> tags. Include the complete statement path down to the element to change.

To confirm that it successfully loaded the configuration as the new candidate configuration, the JUNOScript server returns <rpc-reply></rpc-reply> tags with nothing between them. If it cannot load the configuration, it emits an <xnm:error> tag instead.

Client applications can provide the configuration information to load either as formatted ASCII or tagged with JUNOScript tags. They can also specify the manner in which the JUNOScript server loads the configuration. For instructions, see the following sections:

- Provide Configuration Data as Formatted ASCII or JUNOScript Tags on page 56
- Merge Statements into the Current Configuration on page 56
- Replace (Override) the Entire Current Configuration on page 58
- Replace a Configuration Element on page 58
- Delete a Configuration Element on page 59
- Change a Configuration Element's Activation State on page 62
- Replace a Configuration Element and Change Its Activation State Simultaneously on page 63
- Roll Back to a Previous Configuration on page 65

## **Provide Configuration Data as Formatted ASCII or JUNOScript Tags**

Client applications can use one of two formats when providing configuration data to be loaded into the candidate configuration:

If providing formatted ASCII (the standard format used by the JUNOS CLI show configuration command to display configuration data), set the format attribute on the <load-configuration> tag to the value text. Enclose the configuration data in <configuration-text> rather than <configuration> tags. Format the configuration information to be loaded by using the newline character, tabs and other white space, braces, and square brackets to indicate the hierarchical relationships between the configuration statements.

If providing JUNOScript-tagged information, either set the format attribute on the <load-configuration> tag to the value xml or omit the format attribute completely (the JUNOScript server expects JUNOScript-tagged output by default). Enclose the configuration information in <configuration> tags.

Whichever form of configuration information is provided, enclose the <load-configuration> tag in <rpc> tags. The format attribute can be combined with the action attribute described in “Merge Statements into the Current Configuration” on page 56, “Replace (Override) the Entire Current Configuration” on page 58, and “Replace a Configuration Element” on page 58. For examples of the possible combinations, see those sections.

## **Merge Statements into the Current Configuration**

To combine the statements in the loaded configuration with the current candidate configuration, set the action attribute on the opening <load-configuration> tag to the value merge. This is also the default behavior if there is no action attribute.

```
<load-configuration action="merge">
```

Merging configuration statements is useful when adding a new configuration object or subhierarchy to the configuration. If statements in the loaded configuration conflict with statements in the current candidate configuration, the loaded statements replace the current ones.

As noted in “Provide Configuration Data as Formatted ASCII or JUNOScript Tags” on page 56, client applications can specify the configuration information to load either as formatted ASCII or tagged with JUNOScript tags. In the former case, set the format attribute on the <load-configuration> tag to text.

The following example merges information for a new interface called so-3/0/0 into the [edit interfaces] level of the current candidate configuration. The information is provided in JUNOScript-tagged format (the default).

### Client Application

```
<rpc>
<load-configuration action="merge">
<configuration>
<interfaces>
<interface>
<name>so-3/0/0</name>
<unit>
<name>0</name>
<family>
<inet>
<address>
<name>10.0.0.1/8</name>
</address>
</inet>
</family>
</unit>
</interface>
</interfaces>
</configuration>
</load-configuration>
</rpc>
```

### JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1025

The following example uses formatted ASCII to define the same new interface:

### Client Application

```
<rpc>
<load-configuration action="merge" format="text">
<configuration-text>
interfaces {
    so-3/0/0 {
        unit 0 {
            family inet {
                address 10.0.0.1/8;
            }
        }
    }
</configuration-text>
</load-configuration>
</rpc>
```

### JUNOScript Server

```
<rpc-reply></rpc-reply>
```

T1026

- **Replace (Override) the Entire Current Configuration**

To discard the entire current candidate configuration and replace it with the loaded configuration, set the action attribute on the opening <load-configuration> tag to the value override:

```
<load-configuration action="override">
```

In the following example, the contents of the file /tmp/new.conf (which resides on the router) replaces the entire current candidate configuration. The information in the file is tagged with JUNOScript tags (the default), so the format attribute is not set.

**Client Application**

```
<rpc>
  <load-configuration action="override" url="/tmp/new.conf"/>
</rpc>
```

**JUNOScript Server**

```
<rpc-reply></rpc-reply>
```

T1027

- **Replace a Configuration Element**

To replace a configuration element (hierarchy level or configuration object) in the current configuration, set the action attribute on the opening <load-configuration> tag to the value replace:

```
<load-configuration action="replace">
```

If using JUNOScript tags to define the loaded configuration, include the tags that represent the entire hierarchy down to the element you want to replace. In the opening container tag that represents the element, set the replace attribute to the value replace. Within the element's container tags, include all its child tags.

If using formatted ASCII to define the loaded configuration, include the statements that represent the entire hierarchy down to the element you want to replace. Place the replace: statement above the element's parent statement. Within the container tags for the element, include all relevant child statements.

The following example grants new permissions for the object named operator at the [edit system login class] hierarchy level. The information is provided in JUNOScript-tagged format (the default).

**Client Application**

```
<rpc>
  <load-configuration action="replace">
    <configuration>
      <system>
        <login>
          <class replace="replace">
            <name>operator</name>
            <permissions>configure</permissions>
            <permissions>admin-control</permissions>
          </class>
        </login>
      </system>
    </configuration>
  </load-configuration>
</rpc>
```

**JUNOScript Server**

```
<rpc-reply></rpc-reply>
```

T1028

The following example uses formatted ASCII to make the same change:

**Client Application**

```
<rpc>
  <load-configuration action="replace" format="text">
    <configuration-text>
      system {
        login {
          replace:
            class operator {
              permissions [ configure admin-control ];
            }
        }
      }
    </configuration-text>
  </load-configuration>
</rpc>
```

**JUNOScript Server**

```
<rpc-reply></rpc-reply>
```

T1029

**Delete a Configuration Element**

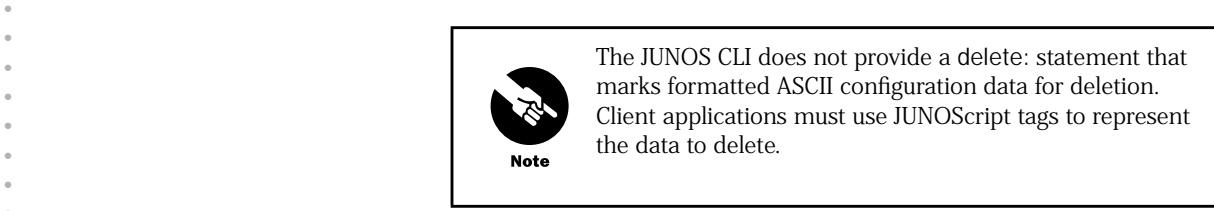
The client application can use the delete attribute to delete several kinds of configuration elements:

[Delete a Hierarchy Level on page 60](#)

[Delete a Configuration Object on page 60](#)

[Delete One or More Values from a Leaf Statement on page 61](#)

[Delete a Fixed-Form Option on page 62](#)



### Delete a Hierarchy Level

To remove a hierarchy level from the configuration hierarchy, set the delete attribute to the value delete on the empty tag that represents the level. Emit <load-configuration> tags that enclose the tags representing the entire statement path down to the level to remove.

The following example removes the [edit protocols ospf] level of the current candidate configuration:

Client Application	JUNOScript Server
<rpc>	
<load-configuration>	
<configuration>	
<protocols>	
<ospf delete="delete"/>	
</protocols>	
</configuration>	
</load-configuration>	
</rpc>	
	<rpc-reply></rpc-reply>

T1030

### Delete a Configuration Object

To delete a single configuration object, set the delete attribute to the value delete on the container tag for that object. Inside the container tag, include the identifier tag only, not any tags that represent other characteristics. Emit <load-configuration> tags that enclose the tags representing the entire statement path down to the object to remove.



The delete attribute is placed on the containing tag rather than the identifier (in the following example, on the <user> tag rather than the <name> tag). The presence of the identifier tag results in the removal of the specified object rather than the removal of the entire hierarchy level represented by the containing tag (in the example, the user account barbara is removed rather than the entire [edit system login user] level).

The following example removes the account for user object barbara from the [edit system login user] level of the current candidate configuration:

Client Application	JUNOScript Server
<pre>&lt;rpc&gt;   &lt;load-configuration&gt;     &lt;configuration&gt;       &lt;system&gt;         &lt;login&gt;           &lt;user delete="delete"&gt;             &lt;name&gt;barbara&lt;/name&gt;           &lt;/user&gt;         &lt;/login&gt;       &lt;/system&gt;     &lt;/configuration&gt;   &lt;/load-configuration&gt; &lt;/rpc&gt;</pre>	<pre>&lt;rpc-reply&gt;&lt;/rpc-reply&gt;</pre>

T1031

### **Delete One or More Values from a Leaf Statement**

To delete one or more values from a leaf statement that accepts multiple values, set the delete attribute to the value delete on the tag for each value. Do not include the tags that represent values to be retained. Emit <load-configuration> tags that enclose the tags representing the entire statement path down to the leaf statement from which to remove values.

The following example removes two of the permissions granted to the user-accounts login class:

Client Application	JUNOScript Server
<pre>&lt;rpc&gt;   &lt;load-configuration&gt;     &lt;configuration&gt;       &lt;system&gt;         &lt;login&gt;           &lt;class&gt;             &lt;name&gt;user-accounts&lt;/name&gt;             &lt;permissions delete="delete"&gt;configure&lt;/permissions&gt;             &lt;permissions delete="delete"&gt;control&lt;/permissions&gt;           &lt;/class&gt;         &lt;/login&gt;       &lt;/system&gt;     &lt;/configuration&gt;   &lt;/load-configuration&gt; &lt;/rpc&gt;</pre>	<pre>&lt;rpc-reply&gt;&lt;/rpc-reply&gt;</pre>

T1032

### **Delete a Fixed-Form Option**

To delete a fixed-form option, set the delete attribute to the value delete on the tag for the option. Emit <load-configuration> tags that enclose the tags representing the entire statement path down to the container tag for the option to remove.

The following example removes the fixed-form disable option at the [edit forwarding-options sampling] hierarchy level:

<b>Client Application</b>	<b>JUNOScript Server</b>
<pre>&lt;rpc&gt;   &lt;load-configuration&gt;     &lt;configuration&gt;       &lt;forwarding-options&gt;         &lt;sampling&gt;           &lt;disable delete="delete"/&gt;         &lt;/sampling&gt;       &lt;/forwarding-options&gt;     &lt;/configuration&gt;   &lt;/load-configuration&gt; &lt;/rpc&gt;</pre>	<pre>&lt;rpc-reply&gt;&lt;/rpc-reply&gt;</pre>

T1033

### **Change a Configuration Element's Activation State**

When a configuration element (hierarchy level or configuration object) is deactivated in the configuration hierarchy, it remains in the candidate configuration but is not activated in the actual configuration when the candidate is later committed.

To use JUNOScript tags to define which element to deactivate, either omit the format attribute from the opening <load-configuration> tag or set it to the value xml. Emit the <configuration> tag next, and then the tags representing the entire statement path down to the element to deactivate. On the tag that represents the element itself, set the inactive attribute to the value inactive. To represent a hierarchy level, emit an empty tag. To represent an object, emit the object's container tag. Inside the container tag enclose only the identifier tag for the object, not any tags that represent other object attributes.

To use formatted ASCII to define the element to deactivate, set the format attribute on the opening <load-configuration> tag to the value text. Emit the <configuration-text> tag next, and within it provide formatted ASCII for all statements in the path down to the element you want to deactivate. Place the inactive: statement immediately before the statement for the element.

To reactivate an element that was previously deactivated, use the preceding instructions and substitute the string active for inactive. Specifically, when loading JUNOScript tags, set the active attribute to the value active on the tag that represents the element to activate. When loading formatted ASCII statements, place the active: statement immediately before the statements to reactivate. With both kinds of notation, the reactivated element is activated in the committed configuration the next time the candidate configuration is committed.

The following example deactivates the [edit protocols isis] level of the current candidate configuration:

<b>Client Application</b>	<b>JUNOScript Server</b>
<pre>&lt;rpc&gt;     &lt;load-configuration&gt;         &lt;configuration&gt;             &lt;protocols&gt;                 &lt;isis inactive="inactive"&gt;                 &lt;/protocols&gt;             &lt;/configuration&gt;         &lt;/load-configuration&gt;     &lt;/rpc&gt;</pre>	<pre>&lt;rpc-reply&gt;&lt;/rpc-reply&gt;</pre>

T1034

## **Replace a Configuration Element and Change Its Activation State Simultaneously**

To replace a configuration element (hierarchy level or configuration object) completely while simultaneously deactivating or reactivating it, set the action attribute on the enclosing `<load-configuration>` tag to the value `replace`. Within the container tags for the element you are replacing, include the tags or statements that represent all the element's characteristics, not just its identifier tag or statement. Finally, combine attributes or statements as follows:

If using JUNOScript tags to replace and deactivate an element, set two attributes on its container tag: the `replace` attribute to the value `replace` and the `inactive` attribute to the value `inactive`. If using formatted ASCII, place the `replace:` statement above the statements to replace and the `inactive:` statement directly in front of the first statement.

If using JUNOScript tags to replace and reactivate an element, set two attributes on its container tag: the `replace` attribute to the value `replace` and the `active` attribute to the value `active`. If using formatted ASCII, place the `replace:` statement above the statements to replace and the `active:` statement directly in front of the first statement.

For more information about completely reconfiguring an element, see “Replace a Configuration Element” on page 58.

- The following example replaces the information in the [edit protocols bgp] level of the current candidate configuration for the group called G3, but also deactivates the group so that it is not activated in the actual configuration when the candidate is committed:

**Client Application**

```
<rpc>
  <load-configuration action="replace">
    <configuration>
      <protocols>
        <bgp>
          <group replace="replace" inactive="inactive">
            <name>G3</name>
            <type>external</type>
            <peer-as>58</peer-as>
            <neighbor>
              <name>10.0.20.1</name>
            </neighbor>
          </group>
        </bgp>
      </protocols>
    </configuration>
  </load-configuration>
</rpc>
```

**JUNOScript Server**

```
<rpc-reply></rpc-reply>
```

T1035

The following example uses formatted ASCII to make the same change:

**Client Application**

```
<rpc>
  <load-configuration action="replace" format="text">
    <configuration-text>
      protocols {
        bgp {
          replace:
            inactive: group G3 {
              type external;
              peer-as 58;
              neighbor 10.0.20.1;
            }
        }
      }
    </configuration-text>
  </load-configuration>
</rpc>
```

**JUNOScript Server**

```
<rpc-reply></rpc-reply>
```

T1036

## **Roll Back to a Previous Configuration**

The router stores a copy of the most recently committed configuration and up to nine previous configurations. To replace the current candidate configuration with a previous one, set the rollback attribute on the empty <load-configuration/> tag to the numerical index for the appropriate previous configuration. The index for the most recently committed configuration is 0 (zero), and the index for the oldest possible stored configuration is 9.

In the following example, the current candidate configuration is replaced by the most recently committed one:

<b>Client Application</b> <pre>&lt;rpc&gt;     &lt;load-configuration rollback="0"/&gt; &lt;/rpc&gt;</pre>	<b>JUNOScript Server</b> <pre>&lt;rpc-reply&gt;&lt;/rpc-reply&gt;</pre>
--	---

T1037

## Verify the Syntactic Correctness of the Candidate Configuration

Before committing the candidate configuration, you might want to confirm that it is syntactically correct. To verify the candidate configuration without actually committing it, enclose the empty <check/> tag in <commit-configuration> tags.

The JUNOScript server encloses its response in <rpc-reply> tags. If the configuration is incorrect, the enclosed <xnm:error> tag explains the nature of the error. The absence of error messages indicates that the operation succeeded.

The following example illustrates the tag sequence when the candidate configuration is syntactically correct:

<b>Client Application</b> <pre>&lt;rpc&gt;     &lt;commit-configuration&gt;         &lt;check/&gt;     &lt;/commit-configuration&gt; &lt;/rpc&gt;</pre>	<b>JUNOScript Server</b> <pre>&lt;rpc-reply&gt;     &lt;output&gt;configuration check succeeds&lt;/output&gt; &lt;/rpc-reply&gt;</pre>
---	--

T1038



Although the JUNOScript server currently indicates success by returning the <output> tag sequence shown in the example, future versions of the JUNOScript API might indicate success with a different string, a different tag, or no tag at all within the <rpc-reply> tag. Client applications must not rely on the presence or content of <output> tags, but instead should interpret the absence of error messages as an indicator of success.

- Commit the Candidate Configuration

To commit the current candidate configuration so that it becomes the active configuration on the router, enclose the empty <commit-configuration/> tag in <rpc> tags. To avoid inadvertently committing changes made by other users or applications, lock the candidate configuration before changing it and emit the <commit-configuration/> tag while the configuration is still locked. (For instructions on locking and changing the candidate configuration, see “Lock the Candidate Configuration” on page 47 and “Change the Candidate Configuration” on page 55.) After committing the configuration, unlock it as described in “Unlock the Candidate Configuration” on page 68.

The JUNOScript server encloses its response in <rpc-reply> tags. If it cannot commit the configuration, the enclosed <xnm:error> tag explains the nature of the error. The most common causes of failure are semantic or syntactic errors in the candidate configuration. The absence of error messages within the <rpc-reply> tags indicates that the operation succeeded.

The following example illustrates the tag sequence when a client application commits the candidate configuration:

**Client Application      JUNOScript Server**

```
<rpc>
  <commit-configuration/>
</rpc>
<rpc-reply>
  <output>commit complete</output>
</rpc-reply>
```

T1039



Although the JUNOScript server currently indicates success by returning the <output> tag sequence shown in the example, future versions of the JUNOScript API might indicate success with a different string, a different tag, or no tag at all within the <rpc-reply> tag. Client applications must not rely on the presence or content of <output> tags, but instead should interpret the absence of error messages as an indicator of success.

### **Commit a Configuration but Require Confirmation**

To commit the current candidate configuration but require an explicit confirmation for the commit to become permanent, emit the empty <confirmed/> tag enclosed in <commit-configuration> tags. If the commit is not confirmed within a certain amount of time (10 minutes by default), the JUNOScript server automatically rolls back to the previously committed configuration. To specify a different number of minutes for the rollback deadline, also emit the <confirm-timeout> tag and enclose an integer value.

The JUNOScript server encloses its response in <rpc-reply> tags. If it cannot commit the configuration, the enclosed <xnm:error> tag explains the nature of the error. The most common causes of commitment failure are semantic or syntactic errors in the candidate configuration. The absence of error messages within the <rpc-reply> tags indicates that the operation succeeded.

The <confirmed/> tag is useful for verifying that a configuration change works correctly and does not prevent management access to the router. If the change prevents access or causes other errors, the automatic rollback to the previous configuration restores access after the rollback deadline passes.

To delay the rollback to a time later than the current rollback deadline, emit the <confirmed/> tag again (enclosed in <commit-configuration> tags) before the deadline passes. Optionally, include the <confirm-timeout> tag to specify how long to delay the next rollback; omit that tag to delay the rollback by the default 10 minutes. The client application can delay the rollback indefinitely by emitting the <confirmed/> tag repeatedly in this way.

To cancel the rollback completely (and commit a configuration permanently), emit one of the following tag sequences before the rollback deadline passes:

The empty <commit-configuration/> tag. The JUNOScript server commits the current candidate configuration immediately and cancels the rollback. If the candidate configuration is still the same as the temporarily committed configuration, this effectively recommits the temporarily committed configuration. The JUNOScript server confirms the commitment by returning an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them.

The empty <check/> tag enclosed in <commit-configuration> tags. The JUNOScript server confirms the syntactic correctness of the candidate configuration and cancels the rollback. The current committed configuration becomes permanently rather than temporarily committed. The JUNOScript server confirms that the candidate configuration is correct by returning an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them.

The following example illustrates the tag sequence when a client application commits the candidate configuration with a rollback deadline of 20 minutes:

#### Client Application

```
<rpc>
  <commit-configuration>
    <confirmed/>
    <confirm-timeout>20</confirm-timeout>
  </commit-configuration>
</rpc>
```

#### JUNOScript Server

```
<rpc-reply>
  <output>commit complete</output>
</rpc-reply>
```

T1040



Although the JUNOScript server currently indicates success by returning the <output> tag sequence shown in the example, future versions of the JUNOScript API might indicate success with a different string, a different tag, or no tag at all within the <rpc-reply> tag. Client applications must not rely on the presence or content of <output> tags, but instead should interpret the absence of error messages as an indicator of success.

- **Unlock the Candidate Configuration**

To unlock the candidate configuration after changing it, committing it, or both, emit the empty <unlock-configuration/> tag enclosed in <rpc> tags. Other applications and users cannot change the candidate configuration until the client application releases the lock. To confirm that it has successfully unlocked the configuration, the JUNOScript server returns an opening <rpc-reply> and closing </rpc-reply> tag with nothing between them. If it cannot unlock the configuration, it returns an <xnm:error> tag instead.

The following example illustrates the tag sequence with which the client application unlocks the configuration:

**Client Application      JUNOScript Server**

```
<rpc>
    <unlock-configuration/>
</rpc>
<rpc-reply></rpc-reply>
```

T1041